Center for Information Systems Research

Massachusetts Institute of Technology
Alfred P. Sloan School of Management
50 Memorial Drive
Cambridge, Massachusetts, 02139
617 253-1000

Contract Number N00039-80-K-0573
Internal Report Number P010-8102-15✓
Deliverable Number 003

12

IMPLEMENTATION AND EVALUATION OF A GRAPH PARTITIONING TECHNIQUE BASED ON A HIGH-DENSITY CLUSTERING MODEL

Technical Report #15

James M. Lattin

February 1981

Principal Investigator:

Professor S.E. Madnick

Prepared for:

Naval Electronics Systems Command

Washington, D.C.

12 68

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. ~~REPORT NUMBER~~ 9 Technical Report #15	2. GOVT ACCESSION NO. AD-A099221	3. RECIPIENT'S CATALOG NUMBER
4. ~~TITLE (and Subtitle)~~ 6 Implementation and Evaluation of a Graph Partitioning Technique based on a High-Density Clustering Model.		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER P010-8102-15
7. ~~AUTHOR(s)~~ 10 James M. Lattin		8. CONTRACT OR GRANT NUMBER(s) 15 N00039-80-K-0573
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Information Systems Research Sloan School of Mgt., M.I.T. Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 11 Feb 81
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Electronics Systems Command		12. REPORT DATE January, 1981
		13. NUMBER OF PAGES 62
14. ~~MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)~~ 14 CISR-P010-8102-15, CISR-TR-15		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release - distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Systematic Design Methodology; High-Density Clustering; Graph Partitioning; Graph Decomposition; Software Architectural Design

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Complex design problems are characterized by a multitude of competing requirements. System designers frequently find the scope of the problem beyond their conceptual abilities, and attempt to cope with this difficulty by decomposing the original design problem into smaller, more manageable subproblems. In the SDM research effort, a systematic approach has been proposed for the decomposition of the set of functional requirements of a design problem into subsets (called subproblems) to form a design structure.

409 590

that will exhibit key characteristics of good design: strong coupling among requirements within subproblems and weak coupling between subproblems.

This report documents the implementation of an efficient graph partitioning technique based on a high-density clustering model. The new method identifies the "high-density regions" in the graph, which are sets of functional requirements exhibiting a relatively high degree of interdependency, and associates them with the different subsets of the design problem.

The new technique, as currently implemented, is applied to several problems from the design literature. The results indicate that the proposed approach gives solutions that are conceptually and intuitively appealing, and that these partitions are consistent with the currently accepted decompositions. Although direct comparison with computational requirements of other partitioning procedures is difficult due to different machine implementations, the empirical evidence suggests that the new method is useful for decomposing design problems too large for the procedures currently in use.

EXECUTIVE SUMMARY

Complex design problems are characterized by a multitude of competing requirements. System designers frequently find the scope of the problem beyond their conceptual abilities, and attempt to cope with this difficulty by decomposing the original design problem into smaller, more manageable subproblems. In the SDM research effort, a systematic approach has been proposed for the decomposition of the set of functional requirements of a design problem into subsets (called subproblems) to form a design structure that will exhibit key characteristics of good design: strong coupling among requirements within subproblems and weak coupling between subproblems.

This report documents the implementation of an efficient graph partitioning technique based on a high-density clustering model. The new method identifies the "high-density regions" in the graph, which are sets of functional requirements exhibiting a relatively high degree of interdependency, and associates them with the different subsets of the design problem.

The new technique, as currently implemented, is applied to several problems from the design literature. The results indicate that the proposed approach gives solutions that are conceptually and intuitively appealing, and that these

partitions are consistent with the currently accepted decompositions. Although direct comparison with computational requirements of other partitioning procedures is difficult due to different machine implementations, the empirical evidence suggests that the new method is useful for decomposing design problems too large for the procedures currently in use.

TABLE OF CONTENTS

1. INTRODUCTION

The Systematic Design Methodology (SDM) was introduced to provide structure for the early stages of the design of complex software systems (Huff [1979]). The third stage of the SDM involves dividing the overall set of functional requirments for a given design problem into subsets for further analysis. In order to limit the conceptual complexity of the design task as much as possible, these subsets are chosen to be mutually exclusive and collectively exhaustive, and to have the property that requirements between subsets are minimally related. The system designer subsequently employs these objectively determined "sub-tasks" and interprets the links between them in order to improve the outcome of the architectural design.

Wong [1980] reviewed several existing techniques for decomposing the graph representation of the design problem, in which each functional requirement of the design problem corresponds to a node and each interdependency to a (weighted) arc in the graph. He found that each of the techniques had some significant shortcoming with respect to the goals of the SDM. In short, they suffered from one or more of the following limitations:

1. The technique required a predetermined number of subgraphs.

2. The technique was incapable of solving problems with a large number of nodes in a reasonable amount of time.
3. The technique operated to optimize some goodness-of-partition criterion that tended to be biased toward extreme solutions (either partitions with many small subgraphs or a few well-balanced ones).

Wong [1980] went on to propose a partitioning technique based on a high-density clustering model on a graph, offering the following motivation for its use:

1. The clustering model identifies the regions of "high-density" (sets of nodes that are highly interconnected) in the graph and thereby suggests to the system designer an appropriate number of subgraphs for the decomposition.
2. The high-density clustering algorithm utilizes a maximum spanning tree formulation, which operates very rapidly on large design graphs (which tend to be relatively sparse).
3. The clustering model does not rely on a goodness-of-partition measure that might tend to favor extreme partitions.

While the report presents some limited examples demonstrating the potential of the model and its conceptual appeal, it does not offer a procedure to determine a partition for the graph based upon the high-density clustering model.

To answer that need, this paper extends and formalizes Wong's partitioning scheme, suggesting an efficient procedure for finding the appropriate partition from the high-density regions of the graph. The approach involves the construction of a maximum spanning tree using only the nearest-neighbor densities, and therefore operates as rapidly as the high-density clustering algorithm. The balance of this paper documents the implementation of the partitioning method, and evaluates its performance with respect to existing decomposition methods used in the design literature. The method is applied to several design problems, including some in the neighborhood of 200 nodes. In each case, the high-density partitioning technique produces a solution in less than a second of computer time that is largely consistent with the accepted "best" existing partition.

The paper follows in several sections. Section 2 reviews the high-density clustering methodology proposed by Wong [1980]. The next three sections outline the implementation of the high-density partitioning technique: section 3 focuses on the calculation of the arc densities Section 4 discusses the maximum spanning tree algorithm (MST) for forming the high-density clusters and for producing the appropriate partition from the high-density regions, and

Section 5 discusses a method for adjusting the partition in order to achieve a minimum number of nodes in each subgraph. Section 6 presents the results of applying the high- density partitioning technique to five design problems from the literature, ranging in size from 22 to 250 nodes, and section 7 concludes the paper with some directions for further investigation in support of the design process.

2. FORMALIZATION OF THE HIGH-DENSITY PARTITIONING TECHNIQUE

2.1 Review of the High-Density Clustering Model

For a given graph G = (N,A), where N is the set of nodes in G and A is the set of arcs connecting the nodes in G, Wong proposes that clusters on a graph are the "densely-connected subgraphs separated from other such subgraphs by relatively few cross links" (Wong [1980], p.8). To identify such clusters, he defines the concept of a density on the arc between any two nodes. For an unweighted graph, this concept of density is operationalized as follows:

d_{ij} = the number of nodes in the neighborhood of node i and node j divided by the number of nodes in either the neighborhood of node i or the neighborhood of node j

that is

$$(1) \qquad d_{ij} = \frac{|N_i \cap N_j|}{|N_i \cup N_j|} \qquad \text{for all } (i,j) \in A$$

where

N_i = the neighborhood of node i,

N_j = the neighborhood of node j, and

$| \cdot |$ gives the cardinality of the enclosed set.

For the purpose of this paper we define the neighborhood of a node i, N_i, to be node i and the set of all nodes in N directly linked to i (i.e. $N_i = \{i\} + \{k \in N | (i,k) \in A\}$).

Depending upon the application, however, it might be desirable to extend the neighborhood concept beyond the nearest-neighbor to two removes; that is, $N_i^{(2)} = \{i\} + \{ k \varepsilon N \mid (j,k) \varepsilon A$ for all $j \varepsilon N_i\}$, where N_i is the neighborhood of node i at one remove (as defined above). Such an extension might provide a more accurate picture of the high-density regions of the graph, but might also involve a prohibitive amount of computation.

For a weighted graph, Wong extends his definition of density by weighting each of the nodes in the numerator: weighting nodes i and j by w_{ij}, and weighting each node k by the average of w_{ik} and w_{kj}. Thus, for a weighted graph, the density measure becomes:

$$(2) \qquad d_{ij} = \frac{2w_{ij} + 1/2 \sum_{k \varepsilon C} (w_{ik} + w_{kj})}{|N_i \cup N_j|}$$

where $C = \{k \varepsilon N \mid (i,k)$ <u>and</u>$(k,j) \varepsilon A\}$.

Because $C = N_i \cap N_j - \{i,j\}$, when $w_{ij} = 1$ for all $(i,j) \varepsilon A$, equation (2) is equivalent to the density calculated for an unweighted graph:

$$d_{ij} = \frac{2 + 1/2 \sum_{k \varepsilon C} (1 + 1)}{|N_i \cup N_j|}$$

$$= \frac{2 + |C|}{|N_i \cup N_j|}$$

$$= \frac{2 + |N_i \cap N_j| - 2}{|N_i \cup N_j|} = \frac{|N_i \cap N_j|}{|N_i \cup N_j|}$$

Thus, it is possible to use the more general equation (2) to calculate the arc densities for both weighted and unweighted graphs. A similar measure is proposed by Andreu[1977], but he does not generalize it to include weighted graphs.

With density thus defined we can examine a graph for sets of nodes where the densities between pairs of nodes are particularly high: these regions of the graph are the high-density clusters. More formally, a high-density cluster at level d* is a subgraph S where S is a maximally connected set whose nodes are connected by links with density at least d*. The value d* thus defines a density contour by delimiting the high-density clusters at level d*.

Figure 1 shows an example of a graph with 13 nodes and 20 unweighted links connecting them; the arc densities are shown in parentheses. The heavy lines in the figure represent a density contour that identifies three high density clusters at level d* = .60: {1,2,3}, {4,5,6,7}, and {9,10,11,12}. In any one of these subgraphs, each node is connected to any other node in the subgraph by some chain of links of density .60 or higher.

Figure 1 here

By likening the density contours on a graph to the altitude contours of a map, the hierarchical (or tree) structure of the high density clusters is apparent. At very high altitudes, only very small regions of the map (such as mountain peaks) are enclosed within the contours at that level. Similarly, at very high densities, there are relatively few groups of small size that are linked at that level. Just as a lower altitude contour encompasses the entire region of the contour above it, so does a lower density contour encompass all the nodes of the contour above it.

Figure 2 shows the 13 node example with three distinct density contours, demonstrating the structure described above. At level $d^* = .50$, the contour drops below the density level of the region separating the second and third subgraphs. These two subgraphs and node 8 thus joins to form one high-density cluster of nine nodes at level $d^* = .50$. As the level reaches $d^* = .40$, node 13 joins this group of nine. Thus, at level $d^* = .40$ the entire graph is included in only two high-density clusters as shown in the figure. Below $d^* =$

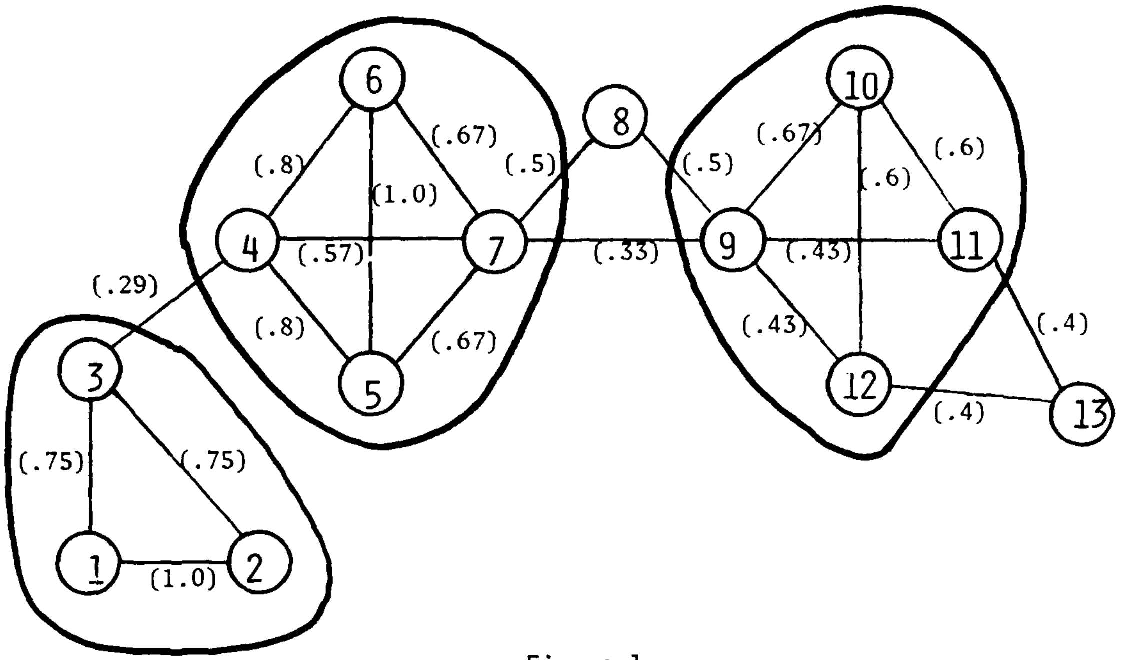


Figure 1

Example showing the three high-density clusters at level d* = .60

.29, which is the region of lowest density, the entire graph forms a single cluster.

Figure 2 here

In his report, Wong introduces the notion of a <u>minimal branching cluster</u>: a maximal high-density cluster at some level d* that has <u>not</u> been formed as a result of the merging of two or more clusters that are distinct at some level higher than d*. Stated more formally, for a high-density cluster S to be a minimal branching cluster, there must exist no more than one high-density cluster within S for every level $d^* \in [0,1.0]$. (That is, a branching cluster S has the property that every cluster properly including S contains some other cluster entirely disjoint from S). Thus, {1,2,3} is a minimal branching cluster, but {4,5,6,7,8,9,10,11,12} is not, because it is formed when {4,5,6,7} and {9,10,11,12} merge. The notion of a minimal branching cluster plays an important conceptual role in determining the appropriate partitioning scheme for a graph. If no more than one branching cluster exists for a particular graph, it might indicate that the graph (and the problem it was designed to represent) does not exhibit a structure appropriate for partitioning.

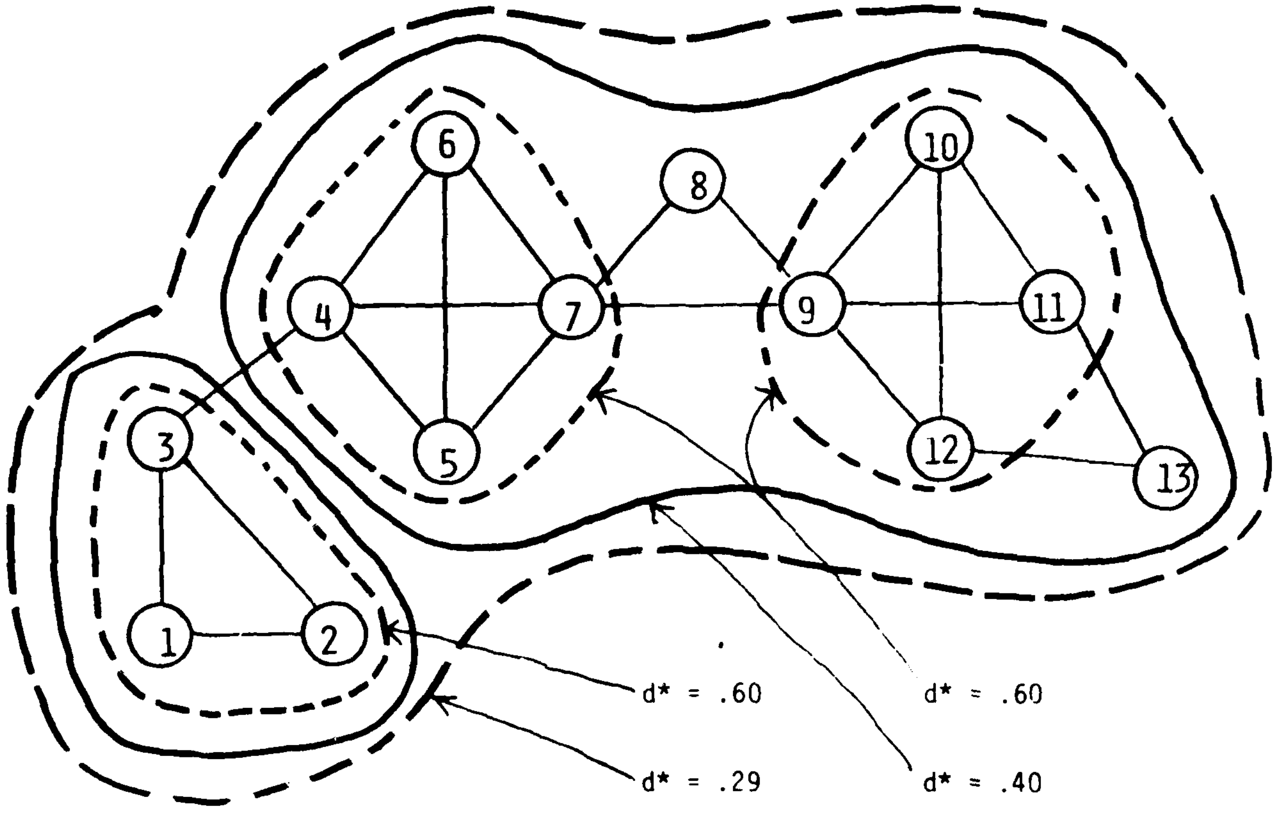


Figure 2

Example demonstrating the tree-like nature of the high-density clusters on a graph

Though the representation of the high-density clustering tree in Figure 2 is highly informative, it is not a convenient representation for a graph with a large number of nodes and arcs. Attempting to draw several hundred nodes and their interconnections is a difficult and frustrating task, and the result ends up obscuring more than it reveals. For a more concise representation, we appeal to the standard clustering tree output shown in Figure 3. (For more information on this form, see Hartigan [1975]).

Figure 3 here

As reported by Wong, the algorithm for finding the clustering tree in Figure 3 is a maximum spanning tree algorithm, which is fully described in Section 4 of this report. The algorithm produces the tree very rapidly, and the output provides a convenient display of information about the node membership of the high-density clusters.

2.2 Choosing a Partition Based on Branching Clusters

What remains to be done is to choose a partition of the graph based on the high-density clustering model. As we have seen, this is not necessarily best done by choosing the high-density clusters at the highest level d* such that all nodes in the graph are included in some cluster. In the 13 node example of Figure 1, the level d* must be at most .40 in

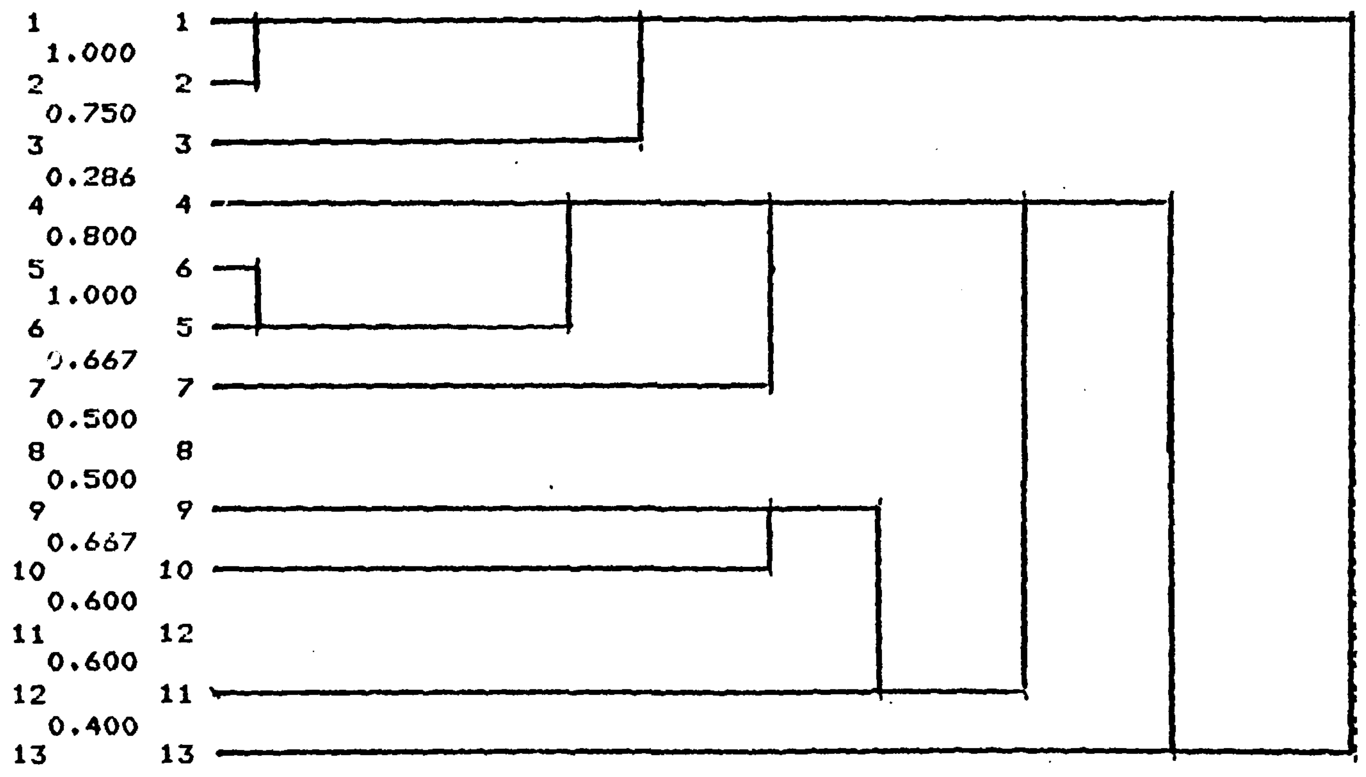


Figure 3

High-density clustering tree for 13 node example order to include each of the nodes in the graph in either one of two clusters: {1,2,3} or {4,5,6,7,8,9,10,11,12,13}. However, the fact that the larger subgraph includes two minimal branching clusters suggests that it should be partitioned further.

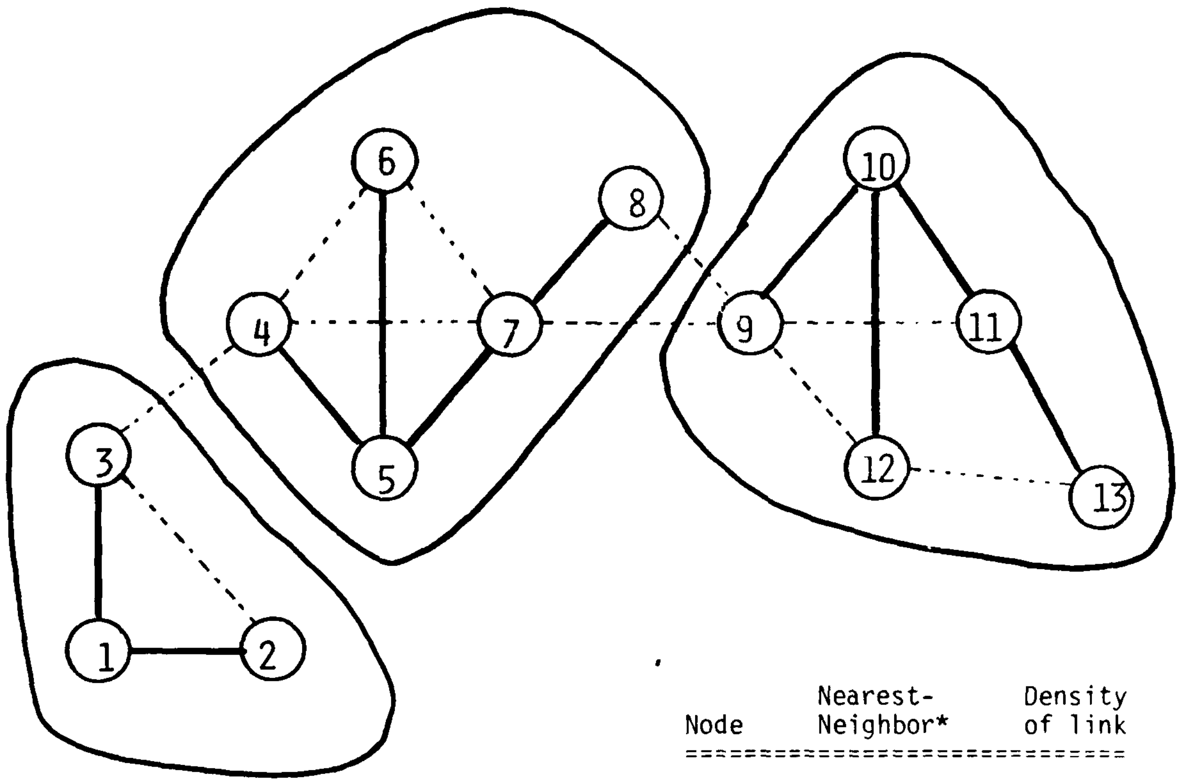
This leads us to focus on the minimal branching clusters to suggest the appropriate partition. In many cases, however, the branching clusters do not indicate a collectively exhaustive set of nodes. In the 13 node example, the three branching clusters include neither node 8 nor node 13. Wong proposed to solve this problem by assigning each "leftover" node to the branching cluster containing the node with the highest density link to the leftover, but suggested no procedure for extracting these branching clusters and determining the assignments.

It is important to note that the clustering tree representation does not provide sufficient information to make these assignments. From Figure 3, we know only that node 13 joins {4,5,6,7,8,9,10,11,12} at level d* = .40, but nothing about its connectivity to either one of the branching clusters {4,5,6,7} or {9,10,11,12}. In order to make such an assignment, we must generate additional information on the nearest neighbor (in terms of densities) of each left-over node. Even with nearest neighbor information, it may be difficult to resolve an assignment. Node 8 in the 13 node graph is clearly a "toss-up" node in that it might just as well be assigned to {4,5,6,7} as {9,10,11,12}. The additional information necessary to place this node correctly must come from the system designer in a later stage of the SDM.

To identify the appropriate partition (i.e. using the minimal branching clusters as subgraphs and assigning leftover nodes to them), we use the following approach, which uses the information provided by the nearest-neighbor links. First, any link between two nodes is defined as a nearest-neighbor link if one node is the nearest-neighbor (in terms of density) of the other. Then, all other links that are not nearest neighbor links are removed from the graph, revealing a mutually exclusive, collectively exhaustive set of connected subgraphs that form the appropriate partition (see figure 4). The computation is done by applying the maximal spanning tree algorithm to the nearest-neighbor densities (described in section 4.3).

Figure 4 here

In order to demonstrate why the approach described above generates a partition that corresponds to the minimal branching clusters within a graph, we appeal to the necessary condition for a branching cluster and utilize the 13 node



——— - indicates a nearest-neighbor link between two nodes

- - - - - - - indicates a link in graph that is not a nearest-neighbor link

Node	Nearest-Neighbor*	Density of link
1	2	(1.0)
2	1	(1.0)
3	1,2	(.75)
4	5	(.80)
5	6	(1.0)
6	5	(1.0)
7	5	(.67)
8	7,9	(.50)
9	10	(.67)
10	9	(.67)
11	10	(.60)
12	10	(.60)
13	11,12	(.40)

* in case of a tie, the nearest-neighbor is the node with the lowest number.

Figure 4

Partition revealed by the nearest-neighbor densities for the 13 node example

example as an illustration. (Figure 4 shows the list of nearest-neighbor links for the 13 node graph, and the partition determined by these nearest-neighbor densities). First, we can assert that any subgraph in a partition generated by the approach outlined above cannot include nodes from more than one branching cluster. If this were the case, there would be a nearest-neighbor link between two nodes of different branching clusters. Clearly, this is impossible, as it would imply that each node is closer (in terms of densities) to the other branching cluster than its own. We can also assert that any subgraph must contain at least one full branching cluster. If there were some subgraph of the partition containing less than the full number of nodes from a given branching cluster, then the omitted node (or nodes) would have no nearest-neighbor link to any node in the branching cluster. Clearly, this would imply that the omitted node would link first to some other node before linking to the rest of the branching cluster, which is contrary to the property of a branching cluster stated above. The partition tree output for the 13 node example is shown in Figure 5.

Figure 5 here

The calculation of arc densities and the formation of the clustering and partition trees are the principal routines

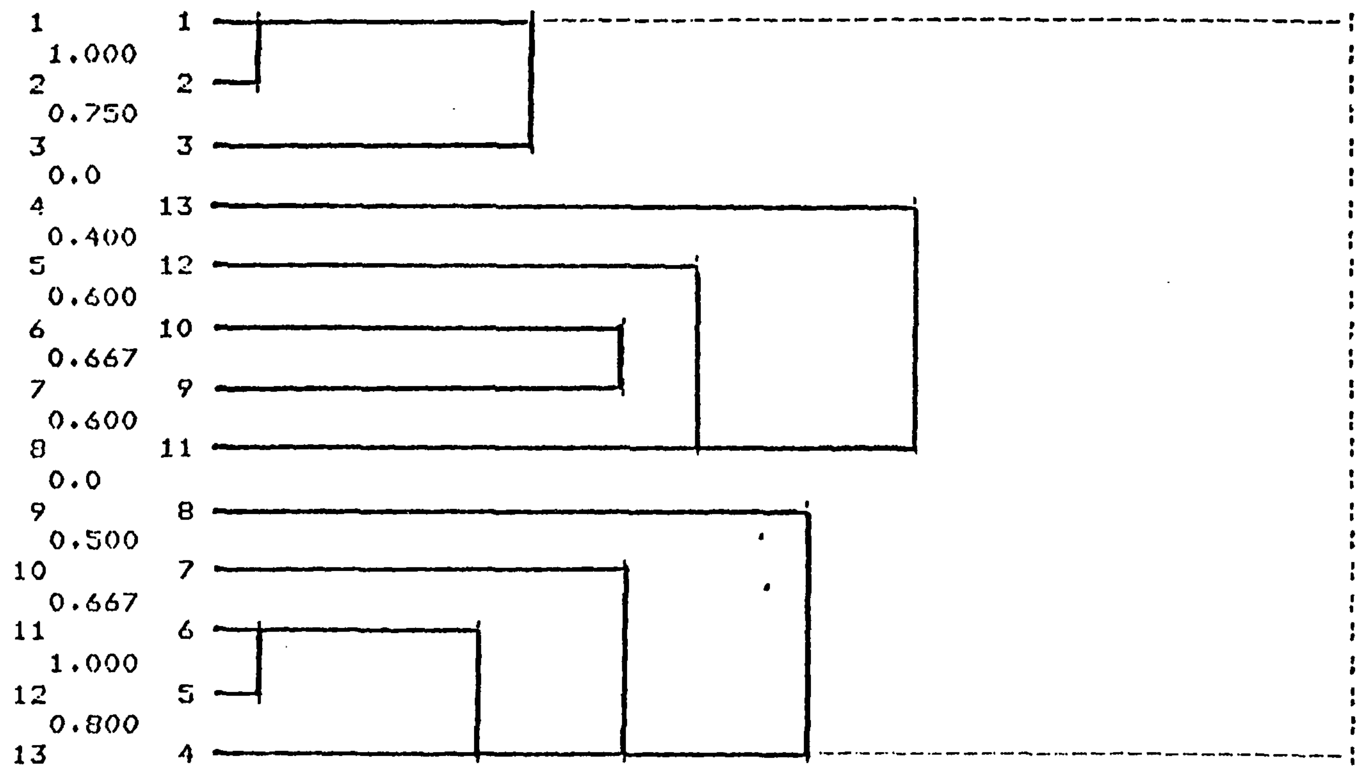


Figure 5

Partition tree for the 13 node example

in the high-density partitioning technique of the SDM. The next sections describe their implementation in FORTRAN IV on an IBM/370, and discuss the order of work required to perform each. The fifth section introduces a fourth routine that adjusts the subgraphs in the partition according to a minimum subgraph size constraint. The result is a heuristic that merges "nearest-neighbor subgraphs" in order to avoid a large number of very small (e.g. two-node) subgraphs.

Armed with a full clustering hierarchy and a suggested partition, we can present the system designer with a great deal of information, which he can use to validate his initial assumptions, look for counter-intuitive results, and further refine his design architecture. The notion of the "sensitivity" of the decomposition results to possible misspecifications or omissions by the system designers is addressed in the concluding section.

3. CALCULATION OF ARC DENSITIES

In these next three sections the computational requirements of a given routine in the high-density partitioning technique are specified in terms of the number of nodes and arcs in the graph. Because we are principally concerned with system design graphs that are typically quite sparse, it will be helpf''l to introducing a measure of the sparsity of the graph $G = (N,A)$. This measure, k, is simply the ratio of the total number of arcs in the graph to the total possible number of arcs.

$$k = \frac{|A|}{|N|(|N| - 1) / 2}$$

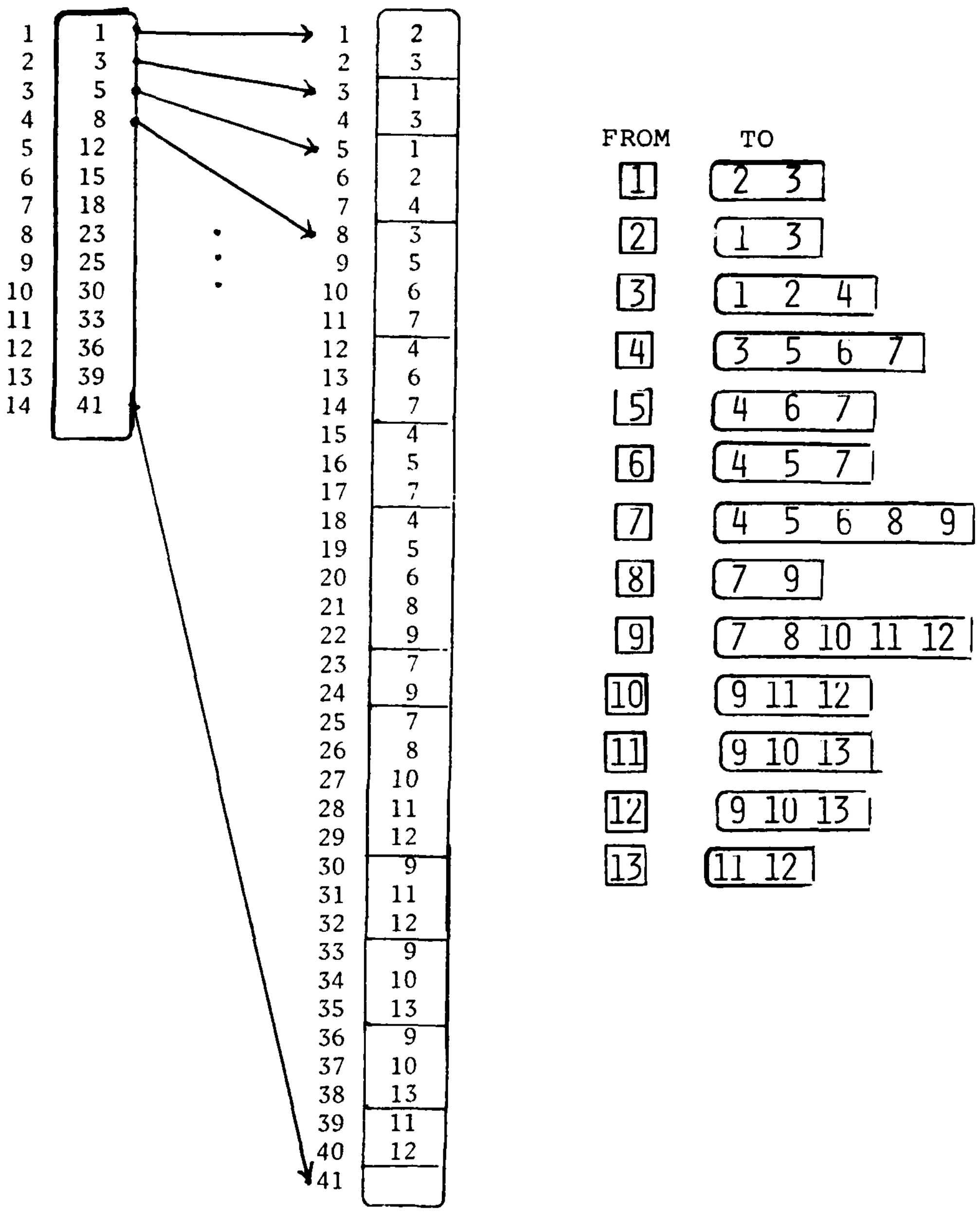
For a ccmplete graph, $k = 1.0$, but for most design graphs, k is substantially smaller. Another useful quantity will be the average number of arcs incident to each node, d, which is equal to $2|A|/|N|$ or $k\,(|N|- 1)$. Again, for a complete graph, $d = |N|- 1$, and so increases linearly with $|N|$. For design graphs, d tends to be much smaller than $|N|$, and may increase as $\sqrt{(|N|)}$ or even $\log(|N|)$. Unfortunately, not enough design problems have been represented as graphs to support such a statement, and so we simply assert that d increases proportionately with $|N|$, albeit a typically small proportion.

3.1 Implementation of Algorithm

The algorithm for calculating the arc densities depends upon the manner in which the graph structure is stored in the computer. Because the graph is typically quite sparse, a node-node adjacency matrix or a node-arc incidence matrix would be a storage-wasteful representation. In order to exploit this sparsity, we choose a type of forward-star representation. (See e.g. Golden and Magnanti [1982]). Figure 6 shows the internal representation for the 13 node example (without arc weights) alongside its conceptual representation. The arc weights are handled in a similar fashion, stored in a vector WEIGHT_LISTS that uses the same INDEX_VECTOR.

Figure 6 here

Note that this storage scheme is not the most parsimonious possible. Because it does not fully exploit the symmetry of the graph, each arc in G is stored twice, thereby requiring $2|A|$ storage locations. Nonetheless, this extra storage space permits a significant computational simplification of the density calculation. The conceptual representation in Figure 6 shows that for each node i in G we have immediate access to the list of nodes in N_i in sorted order. This enables us to determine rapidly the intersection



INTERNAL REPRESENTATION CONCEPTUAL REPRESENTATION

Figure 6

Example representation of
the 13 node graph

and tne union ot the neighborhoods of any two nodes for use in equation (2), repeated below:

$$d_{ij} = \frac{2w_{ij} + 1/2 \sum_{k \varepsilon C} (w_{ik} + w_{kj})}{|N_i \cup N_j|}$$

The algorithm for calculating arc densities is stated below:

STEP 1: Set i = 1

STEP 2: Identify the list of nodes directly connected to node i, and denote it ILIST. This list of values will be a segment of NODE_LIST with indexes from *INDEX_LIST(i) through* INDEX_LIST(i + 1) - 1. Set j = first node in ILIST (at index position INDEX_LIST(i)).

STEP 3: Identify the list of nodes directly connected to node j, and denote it JLIST. This list of values will be a segment of NODE_LIST different from ILIST running from index position INDEX-LIST(j) to INDEX_LIST(j + 1) - 1.

STEP 4: Set NODES_IN_COMMON = 0
Set SUM_WEIGHTS = 0
Systematically compare the contents of ILIST and

JLIST. If a node is common to both, appearing in index position k_i in ILIST and in position k_j in JLIST, do:

a) Set NODES_IN_COMMON = NODES_IN_COMMON + 1

b) Set SUM_WEIGHTS = SUM_WEIGHTS + 1/2 (WEIGHT_LIST(k_i) + WEIGHT_LIST(k_j))

STEP 5: Find w_{ij} in WEIGHT_LIST using the index position of node j in ILIST.

Set UNION = INDEX-LIST(i + 1) - INDEX-LIST(i) + INDEX-LIST(j + 1) - INDEX_LIST(j) - NODES_IN_COMMON

$$\text{Set } d_{ij} = \frac{2w_{ij} + 1/2\ (\text{SUM\_WEIGHTS})}{\text{UNION}}$$

STEP 6: If all nodes connected to node i have been considered (i.e. if d_{ij} has been calculated for all nodes j directly connected to node i), GO TO STEP 7. Otherwise, set j = next node in ILIST and RETURN TO STEP 3.

STEP 7: If $i = |N|$, STOP. Otherwise, set i = i + 1 and RETURN TO STEP 2.

Figure 7 illustrates steps 4 and 5 of the algorithm for node i = 4 and node j = 7.

Figure 7 here

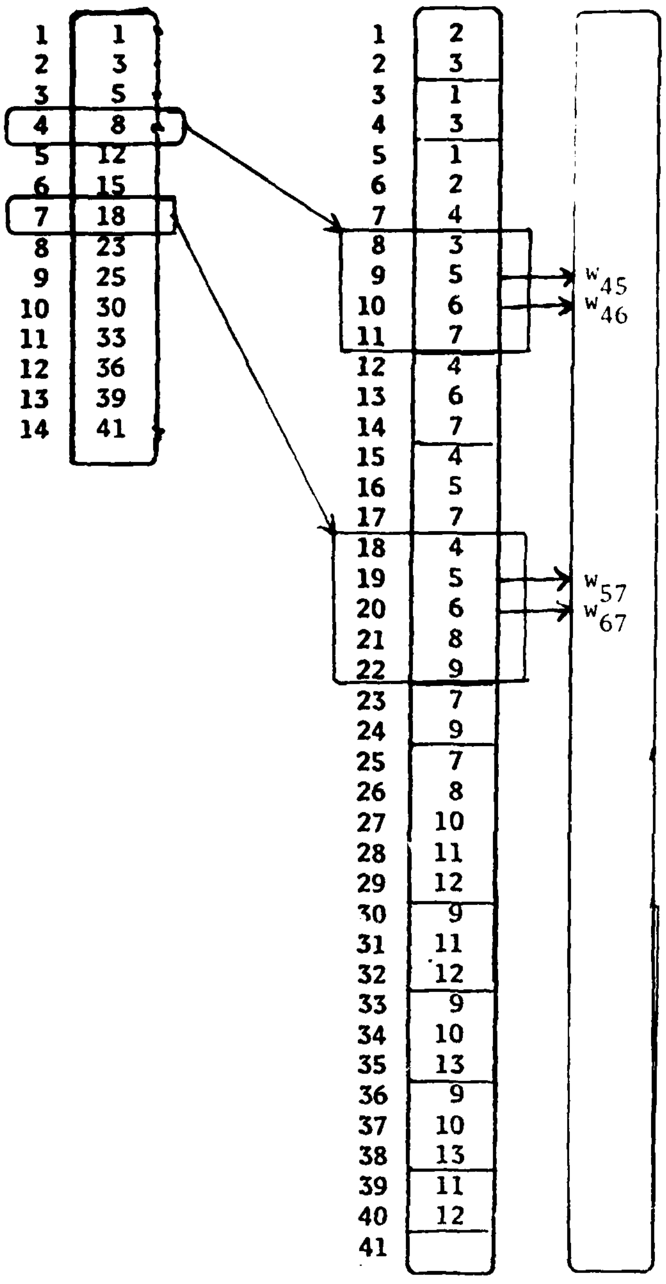
3.2 Computational Requirements

The routine for calculating arc densities shown above is potentially the most time-consuming of the entire high-density partitioning technique. The density calculation must be completed for every arc in the forward star representation, a total of $2|A|$ iterations. Each density calculation involves a systematic comparison of two lists of nodes of average lengths $d = 2|A|/|N|$. Because these lists are sorted, a careful implementation requires only $O(|A|/|N|)$ comparisons. Thus, the total amount of work involved is proportional to $(|A|^2/N)$.

The order of work required to calculate the arc densities is shown below in terms of the average number of arcs incident to each node, d:

$$\begin{aligned} 4|A|^2 / N &= (2|A|/|N|)(2|A|) \\ &= k(|N| - 1)k(|N|)(|N| - 1) \\ &= [k(|N| - 1)][k(|N| - 1)]|N| \\ &= d^2|N| \end{aligned}$$

For a complete graph, where $k = 1.0$, the amount of work required approaches $|N|^3$. However, due to the sparsity of



- NODE I = 4
  ILIST = (3,5,6,7) IN
  INDEX POSITIONS
  8 THROUGH 11.

- NODE J = 7
  JLIST = (4,5,6,8,9) IN
  INDEX POSITIONS
  18 THROUGH 22.

- FIRST NODE IN COMMON IS
  NODE K = 5
  w_{IK} IS IN POSITION 9
  w_{KJ} IS IN POSITION 19

- SECOND NODE IN COMMON IS
  NODE K = 6
  w_{IK} IS IN POSITION 10
  w_{KJ} IS IN POSITION 20

(UNWEIGHTED GRAPH, SO
ALL WEIGHTS = 1.0)

$$d_{4,7} = \frac{2w_{47} + 1/2(w_{45}+w_{57}+w_{46}+w_{67})}{4 + 5 - 2} = \frac{4}{7}$$

Figure 7

Example of density calculation

the graph in most design problems, the order of work is typically much smaller. If d increases as $\sqrt{(|N|)}$, as we speculated above, then the amount of work required is only $|N|^2$; if d increases as $\log|N|$, the requirements are even smaller. Because our experience with design graphs is insufficient to support such claims, we assert only that the computational requirements for the calculation of arc densities are proportional to $|A|^2|/N|$ or $k^2|N|^3$, where k is typically a small fraction.

4. MAXIMUM SPANNING TREE ALGORITHM

4.1 Formation of the High-Density Clustering Tree

The routine for computing the tree of high-density clusters follows quite closely the maximum spanning tree algorithm outlined by Wong [1980] and appearing in Ross [1969] and Hartigan [1975]. The algorithm uses the forward-star representation of the graph stored in INDEX_LIST and NODE_LIST, as well as the vector of arc densities, and forms a type of linked-list vector representation of the clustering tree, a vector of nearest-neighbors, and a vector of nearest-neighbors-within-tree. The vector of nearest-neighbors is later used to form the partitioning tree (described in part 4.2), while the vector of nearest-neighbor-within-tree is used in the heuristic to adjust the size of the subgraphs in the partition (described in section 5).

Figure 8 illustrates the vector representation of the clustering tree for the 13 node example. Arbitrarily, the tree is rooted (or anchored) at node 1 of the graph. The number of the node adjacent to node 1 with the highest density is then stored in TREE_LIST(1) and the density on that arc is stored in the corresponding position in the vector TREE_DENSITIES. In the 13 node example, node 2 has the highest density link to node 1 with $d_{1,2} = 1.0$. Once these values are stored, all the nodes adjacent to node 2 are scanned, and the node closest to either node 1 or node 2 not already in the tree is added by storing that number in TREE_LIST(2). The process continues until all nodes have been added to the tree.

Figure 8 here

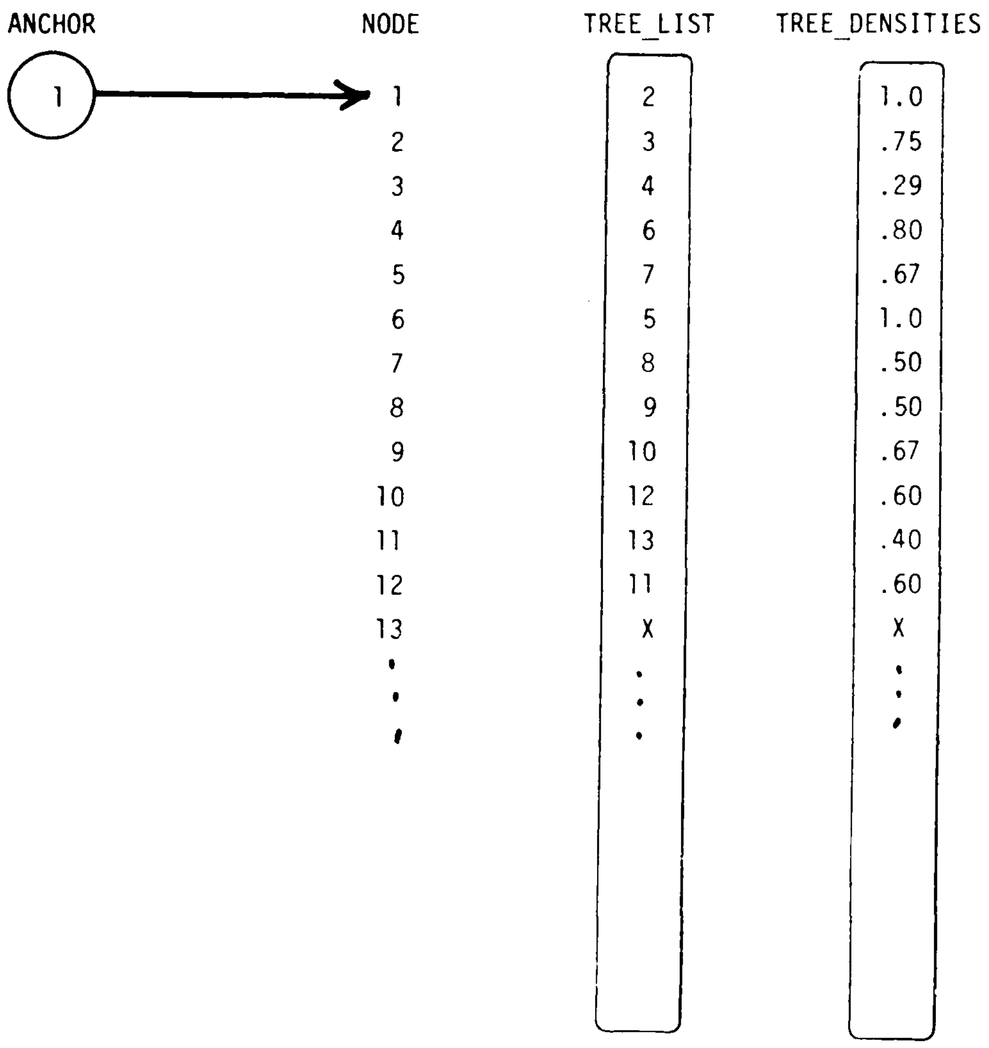
The algorithm for forming the high-density clustering tree as well as the vectors of nearest-neighbors (NN_LIST) and nearest-neighbor-within-tree (NNWT_LIST) is stated below:

```
STEP 1:  Set NN_LIST and NNWT_LIST blank
          LNE (Last Node Entered ) = 1
          Set T = {LNE} where T = set of all nodes in the
          clustering tree.

STEP 2:  Identify the list of nodes adjacent to the LNE, and
          denote it LNE_LIST.

STEP 3:  For each node j in LNE_LIST, do:

            a) If the density on the arc between node j and
               LNE is greater than the density between node j
               and any node scanned so far (i.e. greater
               than the density on the link between j and
               NN_LIST(j)), then set NN_LIST(j) = LNE.
```



ANCHOR	NODE	TREE_LIST	TREE_DENSITIES
1	1	2	1.0
	2	3	.75
	3	4	.29
	4	6	.80
	5	7	.67
	6	5	1.0
	7	8	.50
	8	9	.50
	9	10	.67
	10	12	.60
	11	13	.40
	12	11	.60
	13	X	X
	⋮	⋮	⋮

Figure 8

Vector representation of clustering tree

b) If node $j \notin T$ and the density between node j and LNE is greater than between node j and any node scanned so far (i.e. greater than the density on the link between j and NNWT_LIST(j)) then set NNWT_LIST(j) = LNE.

c) If node $j \notin T$ and the density on the arc between node j and LNE is greater than any previous link in the clustering tree to node j (i.e. greater than TREE_DENSITIES(j)), then set TREE_DENSITIES(j) = $d_{j,LNE}$.

STEP 4: Search TREE_DENSITIES for the highest entry among nodes $\notin$ T and set NNE(Next Node Entered) equal to number of the node for that entry.

STEP 5: Set TREE_LIST(LNE) = NNE

Set T = T + {LNE}

If T = N STOP. Otherwise, RETURN TO STEP 2.

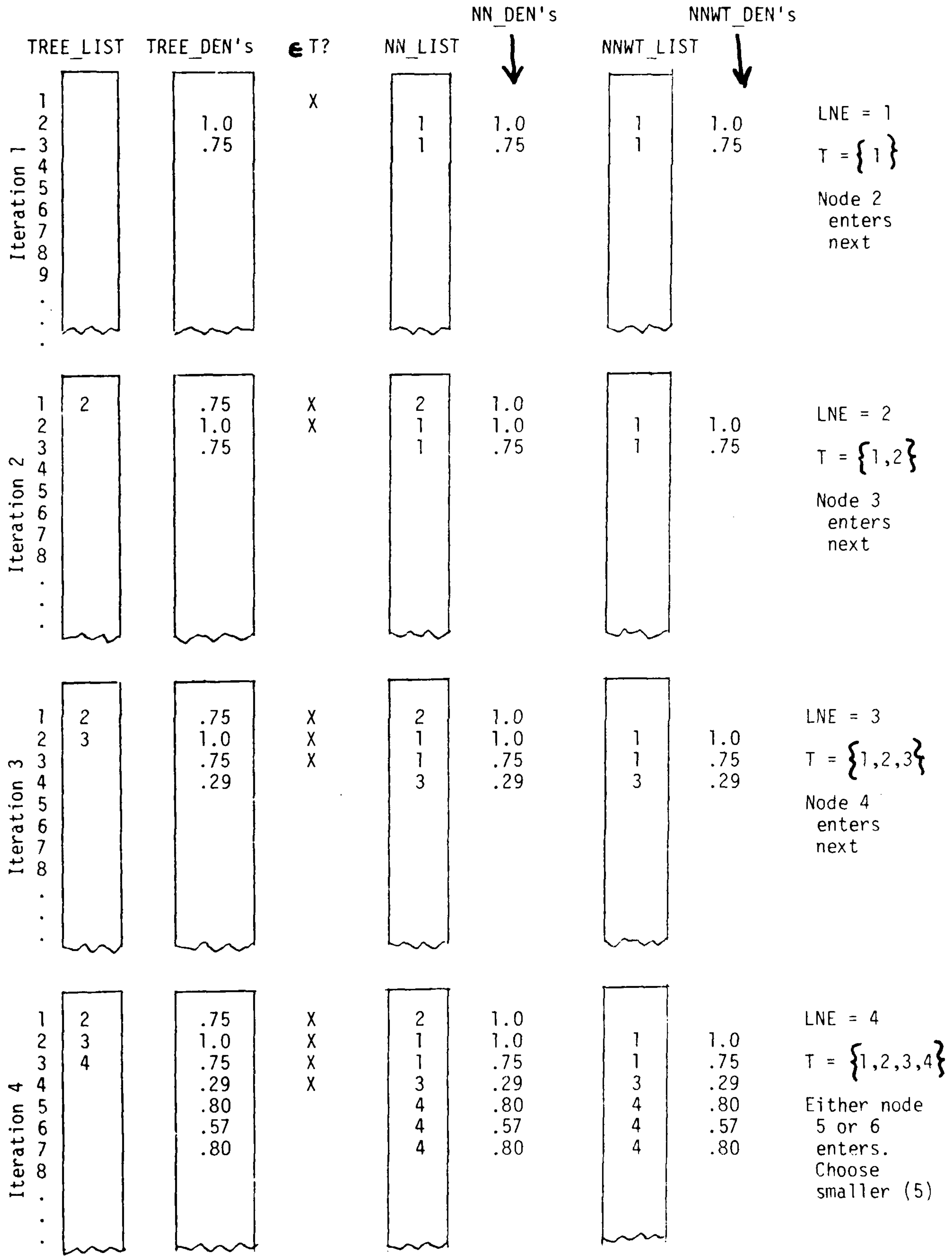
Figure 9 shows "snapshots" of the vectors being built by the clustering tree algorithm. Each one of the four snapshots is taken at an iteration of the algorithm just before STEP 4, when the vector TREE_DENSITIES is searched to determine the next node to enter the clustering tree. In the fourth iteration, the last shown in the figure, node 4 enters the tree. To indicate this, the number 4 is stored in TREE_LIST at the position of the last node entered, which is

3, and an 'X' appears alongside position 4. Node 4 now becomes LNE, and the list of adjacent nodes {3,4,5,6,7} is considered in STEP 3 of the algorithm. Node 3, which is already in the tree, clearly does not have a higher density link to node 4 (LNE) than to node 1 (compare $d_{3,4} = .29$ to NNWT_DENSITIES(3) = .75) so no change occurs. Nodes 5,6, and 7, however, have not yet been considered, and so the corresponding densities are entered as shown in the fourth snapshot. The algorithm continues in this manner through $|N| - 1$ iterations, at which point all the vectors in Figure 9 are complete.

Figure 9 here

4.2 Computational Requirements

The maximum spanning tree algorithm outlined above is very rapid. The algorithm involves a total of $|N| - 1$ iterations. Each iteration must consider the list of arcs adjacent to the last node entering the tree, which is of average length d. Thus, the amount of work required is proportional to $d(|N| - 1)$ or roughly $2|A|$. Regardless of the nature of the graph, this is never more than $|N|^2$, and for the design graphs in question it is significantly less. While the order of work might be as small as $|N|^{3/2}$ or even $|N|\log|N|$, we assert only that the computational requirements are proportional to $|A|$ or $k|N|^2$, where k is typically a



Iteration 1

	TREE_LIST	TREE_DEN's	∈ T?	NN_LIST	NN_DEN's	NNWT_LIST	NNWT_DEN's
1			X				
2		1.0		1	1.0	1	1.0
3		.75		1	.75	1	.75
4							
5							
6							
7							
8							
9							
. . .							

LNE = 1

$T = \{1\}$

Node 2 enters next

Iteration 2

	TREE_LIST	TREE_DEN's	∈ T?	NN_LIST	NN_DEN's	NNWT_LIST	NNWT_DEN's
1	2	.75	X	2	1.0		
2		1.0	X	1	1.0	1	1.0
3		.75		1	.75	1	.75
4							
5							
6							
7							
8							
. . .							

LNE = 2

$T = \{1,2\}$

Node 3 enters next

Iteration 3

	TREE_LIST	TREE_DEN's	∈ T?	NN_LIST	NN_DEN's	NNWT_LIST	NNWT_DEN's
1	2	.75	X	2	1.0		
2	3	1.0	X	1	1.0	1	1.0
3		.75	X	1	.75	1	.75
4		.29		3	.29	3	.29
5							
6							
7							
8							
. . .							

LNE = 3

$T = \{1,2,3\}$

Node 4 enters next

Iteration 4

	TREE_LIST	TREE_DEN's	∈ T?	NN_LIST	NN_DEN's	NNWT_LIST	NNWT_DEN's
1	2	.75	X	2	1.0		
2	3	1.0	X	1	1.0	1	1.0
3	4	.75	X	1	.75	1	.75
4		.29	X	3	.29	3	.29
5		.80		4	.80	4	.80
6		.57		4	.57	4	.57
7		.80		4	.80	4	.80
8							
. . .							

LNE = 4

$T = \{1,2,3,4\}$

Either node 5 or 6 enters. Choose smaller (5)

Figure 9

small fraction.

4.3 Formation of the Partition Tree from Nearest Neighbor Densities

The same algorithm that generates the high-density clustering tree can also be used to produce the partition tree by applying the nearest-neighbor densities as described above in Section 2. Figure 10 shows the nearest-neighbor densities for the 13 node example in the form used by the maximum spanning tree algorithm. Once these densities have been put into the form shown in the figure, the computational requirements for forming the partition tree are the same as those stated for the high-density clustering tree.

Figure 10 here

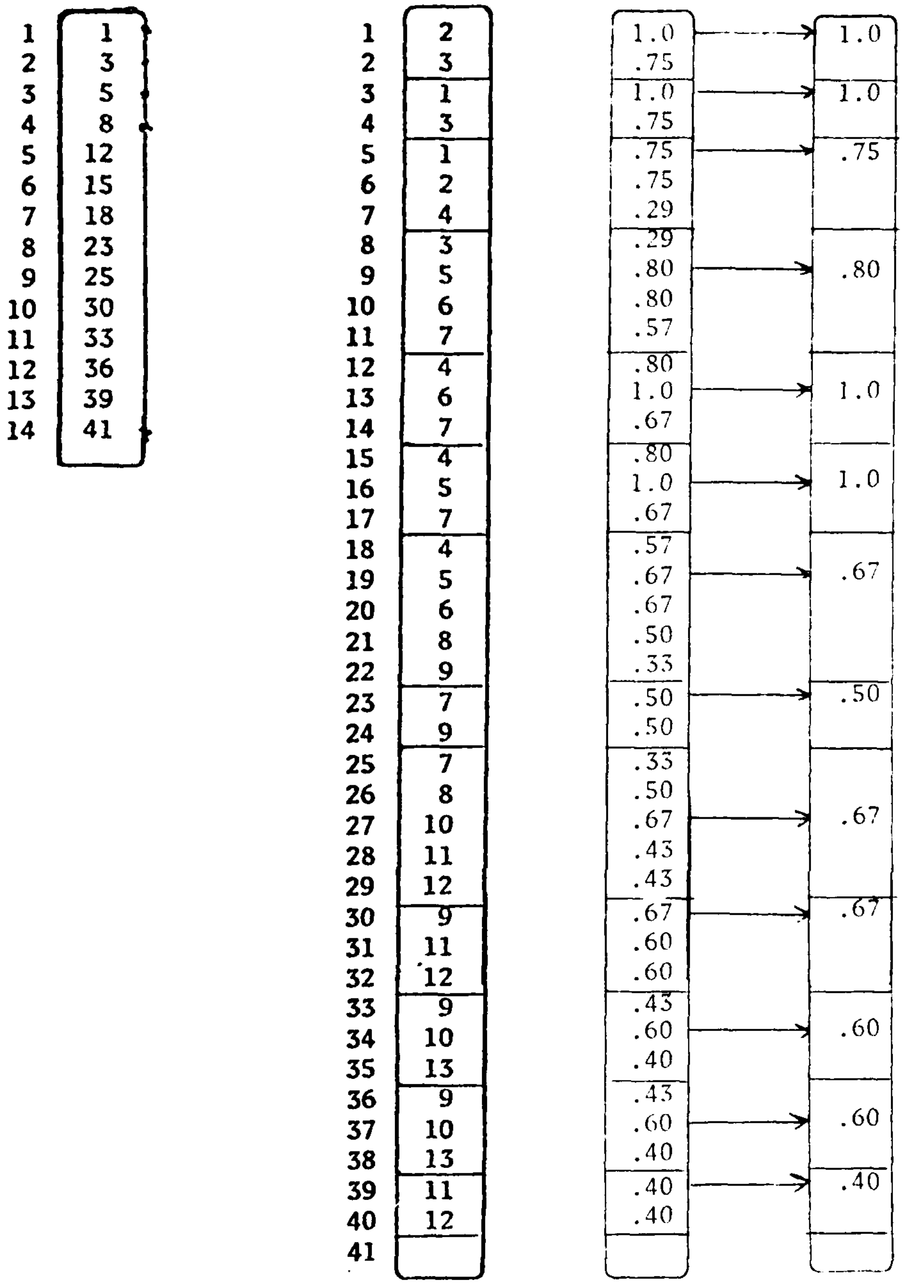


Figure 10

Representation of nearest-neighbor densities

5. ADJUSTMENT OF SUBGRAPH SIZE

5.1 Implementation of Heuristic

When the number of nodes in a graph is substantial, the number of subgraphs in the resulting partition may also be quite large. It is often desirable to reduce the number of small subgraphs in a given partition in order to simplify the task of interpreting the interactions between all pairs of subgraphs. We choose to do this by merging "nearest-neighbor-subgraphs" until each subgraph or modified subgraph meets a minimum size constraint. We do this heuristically, examining each subgraph to see that it meets the minimum specified size and, if it does not, choosing a "central" node and merging the subgraph with the next subgraph closest to this central node. This merging process is not perfect, as the choice of a central node is not always well defined. Nonetheless, the method is quite fast, and in the end the system designer has recourse to the partition tree supplied by the heuristic.

Figure 11 here

The partition tree for the 13 node graph before subgraph modification is shown in Figure 11. This information (along with the clustering tree and the nearest-neighbor lists) is used in the modification heuristic outlined below:

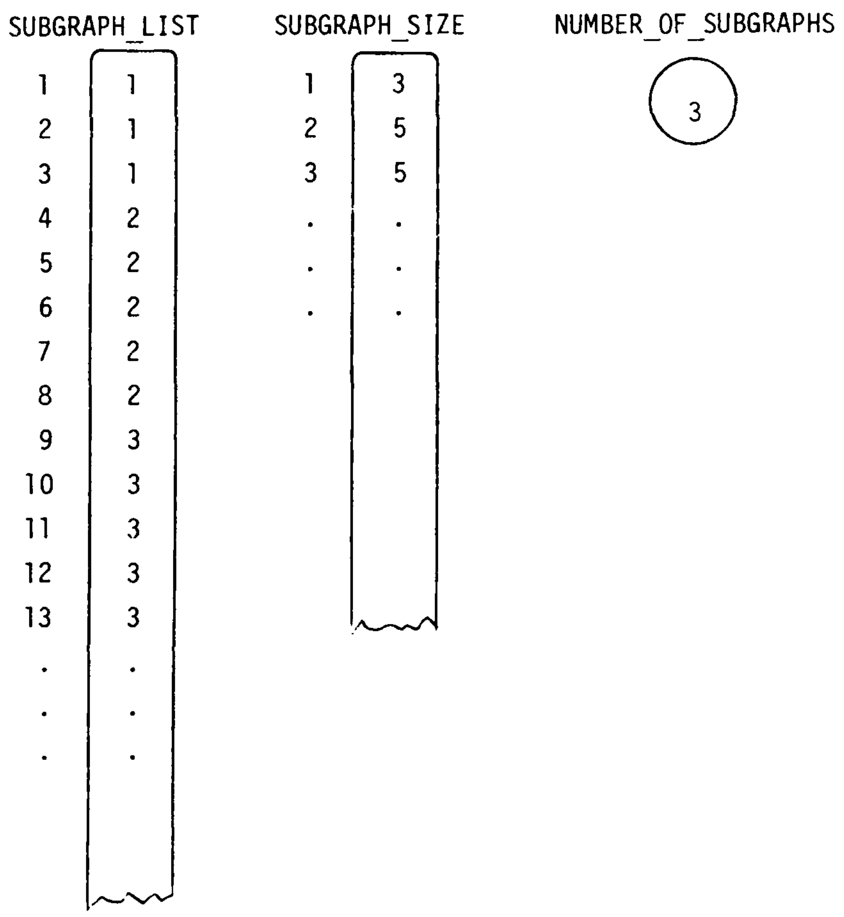


Figure 11

Partition formed by maximum spanning tree of nearest-neighbors

STEP 1: Let M be the minimum permitted size of a subgraph in the partition. Set NUM equal to the number of subgraphs in the partition. Set CURRENT_SUBGRAPH = 1.

STEP 2: Check the subgraph size of CURRENT_SUBGRAPH. If SUBGRAPH_SIZE(CURRENT_SUBGRAPH) $\geq$ M, GO TO STEP 6. Otherwise continue.

STEP 3: Locate the first node in SUBGRAPH_LIST that is an element of CURRENT_SUBGRAPH, and subsequently locate the position of this node in the partition tree.

STEP 4: From the current position in the partition tree read "up" and "down" within CURRENT_SUBGRAPH in order to find a node adjacent to the highest density link in the subgraph. Denote this node CENTRAL_NODE and subsequently locate its position in the clustering tree.

STEP 5: From the position of CENTRAL_NODE in the clustering tree, do:

a) Read "up" in the tree until a node from a subgraph other than CURRENT_SUBGRAPH is found, and note the density separating this node from CURRENT_SUBGRAPH.

b) Read "down" in the tree in the same manner as in (a) above.

If the density found in (a) is greater, then merge CURRENT_SUBGRAPH of the node which is the nearest-neighbor of CENTRAL_NODE within the tree (i.e. NNWT_LIST(CENTRAL NODE)).

If the density found in (b) is greater, then merge CURRENT_SUBGRAPH with the subgraph of the node reached by reading "down" in the clustering tree.

Accomplish the merge by reading through SUBGRAPH_LIST and replacing every occurrence of CURRENT_SUBGRAPH with the number of the new subgraph. Then, set SUBGRAPH_SIZE (new subgraph) = SUBGRAPH_SIZE (new subgraph)+ SUBGRAPH_SIZE (CURRENT_SUBGRAPH). Set NUM = NUM - 1.

STEP 6: If CURRENT_SUBGRAPH is the last in the partition, STOP. Otherwise, set CURRENT_SUBGRAPH = CURRENT_SUBGRAPH + 1 and RETURN to STEP 2.

In the current implementation of this routine, we arbitrarily establish the minimum subgraph size M as follows:

$$M = \lfloor |N| / 10 \rfloor + 2$$

where $\lfloor \alpha \rfloor$ equals the greatest integer part of α. An interactive subroutine enables the system designer to vary the size of M in order to achieve different modifications of the partition.

5.1 Computational Requirements

On average, the heuristic presented above is quite rapid, as it needs only modify the subgraphs which have fewer than M nodes. In the worst case, there might be as many as $S = |N| / 2$ subgraphs, each with two nodes. If it turns out that each of the subgraphs 1,2,3...,S-1 merge with subgraph S, then the heurstic might involve as many as $(|N| / 2) - 1$ iterations. Each iteration involves searching three lists of dimension $|N|$, and modifying a fourth list also of dimension $|N|$. Thus, in the worst case, the amount of work required to modify the partition is at most proportional to $|N|^2$. Experience with several real design graphs suggests, however, that the number of subgraphs modified is quite small, and the computational requirements are minimal.

6. PERFORMANCE OF THE PARTITIONING TECHNIQUE ON REAL DESIGN GRAPHS

The following section presents the results of applying the High-Density Partitioning Technique to five real design problems taken from the graph decomposition literature, ranging in size from 22 to 77 functional requirements. For each one of the five design graphs, we present the existing decomposition and the high-density partition and note the significant similarities and differences. We also present computational results from the new method on graphs of up to 250 nodes, and, when available, the results from other decomposition techniques.

We conclude from our comparison that the new technique is a useful heuristic that enables systems designers to better focus on the global properties of their design specifications. The high-density partition provides the designer with information with which he can check his initial assumptions and further refine his design architecture. In some cases, the high-density clustering solution reveals opportunities in the definition of the design sub-tasks clearly missed by the other methods employed. In all cases, the new technique renders a solution well in the "ballpark" of accepted decompositions, and does so faster than other existing methods.

6.1 Database Management System: Unweighted Interdependencies

Andreu and Madnick [1977] outlined the design of a database management system (DBMS). They listed 22 functional requirements, (e.g. minimal data redundancy, rapid data reference, and unambiguous query language) and 39 unweighted interdependencies in their design problem. Andreu [1978] used a variety of hierarchical clustering techniques and an iterative partitioning approach to produce several decompositions of the design graph, all of which were identical to the partition shown in Figure 12.

The decomposition produced by the high-density partitioning technique is shown in Figure 12 adjacent to the solution proposed by Andreu, revealing the similarity between the two. The sets of circled nodes in the figure represent the original subgraphs of the unmodified partition; the numbered subgraphs illustrate the result of merging the original subgraphs to meet a minimum size constraint of $M = \lfloor 22/10 \rfloor + 2 = 4$. The only difference between the two is that in the high-density decomposition {1,2,3,4,5,6,9,21} is a single subgraph, while in the decomposition proposed by Andreu it is two subgraphs: {1,2,3,5} and {6,9,21}. Nonetheless, from the output provided by the high-density partitioning routine, the designer can see that {1,2,3,5,6,9,21} was formed as a result of merging two smaller subgraphs to meet the minimum size constraint. In fact, when M = 3, the two solutions are identical.

Figure 12 here

1	1,2,3,5
2	6,9,21
3	7,13,14,15
4	4,16,17,18,22
5	8,10,11,12,19,20

Partition proposed by Andreu [1978]

1	1,2,3,5	6,9,21
2	7,13,14,15	
3	4,16,17,18,22	
4	8,12	10,11,19,20

Solution generated by high-density partitioning technique (for M = 4)

Figure 12

Andreu and Madnick's [1977] 22 node (unweighted) Database Management System design problem

6.2 Database Management System: Weighted Interdependencies

Huff and Madnick [1978] modified the 22-node DBMS problem by weighting the interdependencies between the functional requirements of the design structure. They chose the following arbitrary assignment:

STRONG INTERDEPENDENCY = [illegible]

AVERAGE INTERDEPENDENCY = .50

WEAK INTERDEPENDENCY = [illegible]

Huff [1979] like Andreu, also used a variety of clustering techniques and an interchange partitioning algorithm (INTERCHANGE) to decompose the DBMS design graph. Huff chose as his solution the partition with the highest "measure of merit," a measure chosen to reflect the extent to which the partition meets the key characteristics of good design. Huff's solution is shown in Figure 13.

The high density partition for the weighted 22-node DBMS example, shown also in Figure 13, differs from the partition proposed by Huff in only one respect: Huff includes node 21 in {4,16,17,18,21,22} while the high-density solution includes node 21 in {1,2,3,5,6,21}. According to the design graph, shown in Figure 14, node 21 (a requirement calling for minimal data redundancy) is related to two other requirements in each of two subgraphs. In Huff's partition, node 21 has links within the subgraph to nodes 4 (calling for algorithmic relationships among data items) and 22 (minimal storage costs), each with average weight. In the high-density partition, node 21 has links within the subgraph to 1 (calling for multiple logical organizations of the database) with the average weight, but to 6 (unique physical organization) only weakly. Because Huff's measure of merit rewards strength of coupling within a subgraph and penalizes links between subgraphs, Huff's measure is higher if node 21 joins (4,16,17,18,...), thus preserving the two links of average weight.

Figure 13 here

The high-density model calls for a different result. Although the weight on the arc between node 6 and 21 is only weak, the density on that arc [illegible] is the highest on any arc adjacent to node 21. This is by virtue of the fact that node 6 and node 21 are linked primarily to common nodes. Thus, in the high density partition, node 21 joins to [illegible].

Figure 14 here

Either approach seems to have a legitimate argument for its treatment of node 21 (minimal data redundancy). In Huff's solution, 21 joins a group of nodes representing data

1	1,2,3,5,6	1	1,2,3,5,6,21
2	4,16,17,18,21,22	2	4,16,17,18,22
3	8,9,10,11,12,19,20	3	8,12 9,10,11,19,20
4	7,13,14,15	4	7,13,14,15
	Partition proposed by Huff [1979]		Solution generated by the high-density partitioning technique (M = 4)

Figure 13

Huff and Madnick's [1978] adaptation of Andreu's 22 node DBMS design problem

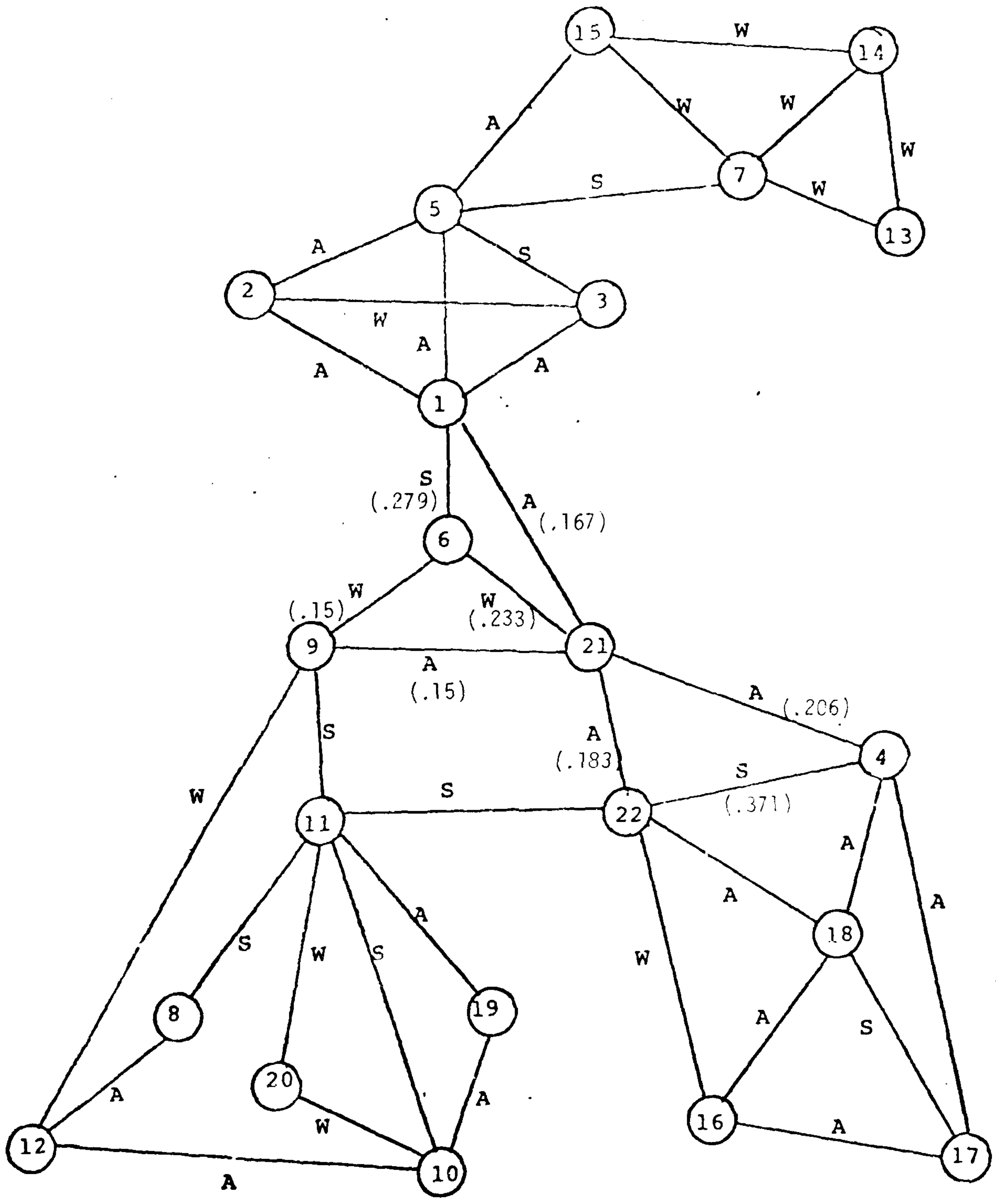


Figure 14

Huff's [1979] presentation of the 22 node DBMS problem storage characteristics and requirements; in the high-density partition node 21 is included with the nodes representing the requirements for logical organization and relationships among data. The final decision on the appropriate placement of such a functional requirement within a sub-task must rest with the system designer.

6.3 The Airport Design Problem

McCormick *et. al.* [1972] used a 27-node airport design problem to illustrate the effectiveness of their Bond Energy Algorithm (BEA). The results are shown in Figure 15. The BEA does not suggest mutually exclusive, collectively exhaustive sets of functional requirements, but rather identifies overlapping "clumps" of interrelated requirements. The system designer must subsequently decide how best to divide these requirements into sub-tasks. McCormick reports that the BEA required approximately 60 CPU seconds for a CDC 1604 computer to reach solution for the 27-node graph. The high-density partition for the airport design problem is shown next to the Bond Energy solution in Figure 15. The subgraphs are ordered from top to bottom so as to correspond as closely as possible to McCormick's presentation and facilitate comparison.

Figure 15 here

Though both solutions present an intuitively appealing decomposition of the design problem there are some significant differences between the two. McCormick inclues node 18 (Concessions) with 25 (refuse removal) and 27 (aircraft service on apron). In the high-density partition node 18 joins a group of nodes representing passenger information and check-ins. Examining the connectivity of node 18, we find that it has links to two of the nodes in the subgraph suggested by the high-density partitioning technique, while it has one double-weighted link to a node in the subgraph suggested by the *Bond Energy Approach.* In general, the bond energy objective function is best improved by preserving the weighted links at the expense of severing a number of smaller links.

In another illustration of the difference between the two proposed solutions, the BEA joins nodes 11 and 12 (service area and parking lots for rental cars) together in a subgraph with node [illegible] (rental desk), while the high- density solution includes nod[illegible] 1 and 12 with 7 (close parking lots), 9 (main access road), an 10 (circulation roads). It turns out that only node 12 is linked to node 19; however, nodes 11 an 12 have three links to {7,9,10}. In general, it

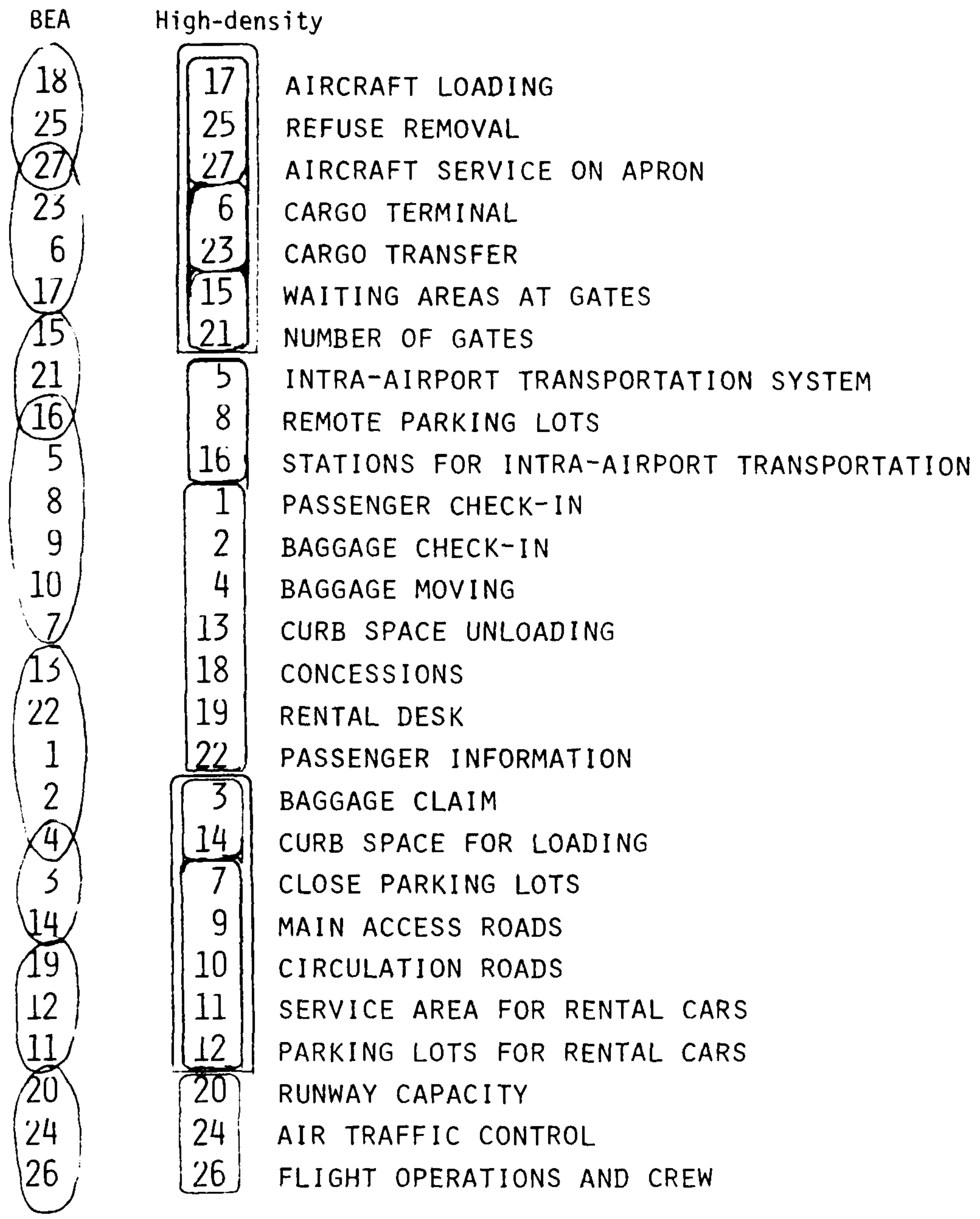


Figure 15

McCormick <u>et</u>. <u>al</u>. [1972] solution to
the Airport Design Problem

seems that the diagonalization process used by the BEA may not always be appropriate for positioning pairs or small groups of heavily-linked nodes as part of a larger subgraph.

The high-density technique required about 0.15 CPU seconds on an IBM/370 168 to read the data, calculate the arc densities, compute the clustering and partition trees and adjust the partition for the minimum subgraph size constraint. This is a 400-fold improvement over the computational requirements of the BEA for the same problem though it is not strictly legitimate to compare performance results across machines. However, McCormick reports that for a graph of $|N|$ items the number of operations in $|N|^3$, taking to account of the sparsity of the graph. For the high-density technique, the number of operations increases as $k^2|N|^3$, where $k^2 = .075$ for the airport problem. For sparse design graphs with very large $|N|$, McCormick's BEA is at a significant computational disadvantage.

6.4 The Design of a Printed Circuit Board (PCB) Test Facility

Tung [1980] outlined 69 functional requirements for the design of a PCB Test Facility. Due to the rather technical nature of the test facility, the reader is referred to Tung's report for explanations of the requirements and interdependencies. The decomposition proposed by Tung, based on a time-consuming ad hoc search to determine the best value for Huff's measure of merit (see section 6.2 above) appears in Figure 16 below.

The high-density partition of the PCB Test Facility design graph is shown in Figure 16 adjacent to the solution proposed by Tung. Due to the significant size of this problem, it is difficult to accurately assess the similarity of the two results by simply scanning them. At the lowest level of grouping presented in the high-density partition (indicated by the circled groups of nodes in the picture), these groups are almost always left intact in Tung's proposed solution. The exceptions are the high-density subgraphs {16,26,28,29,34,41,42,43} and {46,49,56,58,64,65,66,67}.

Figure 16 here

At a somewhat higher level of grouping, there are some disagreements between the two solutions, due to the fact that partitioning a graph to somehow minimize the collective weight of the arcs severed is not always consistent with partitioning a graph through the regions of lowest density. A good illustration is the appropriate placement of the completely connected group {44,50,54}. Tung joins this group with the nodes {37,38,39,40,45,47,48,53,55,59} principally because all three are linked to node 45 which is in turn linked to many of the nodes in the same group. The high-density solution joins this group to nodes {27,30,32,

1	1,2,7,10,11,12,13,14	1	$\boxed{1,2,7,10,11,14}$ $\boxed{3,4}$ $\boxed{8,9}$ $\boxed{12,13}$ $\boxed{45,53,55}$
2	5,6,8,9,15,16,17,18,19,20 21,22,23,24,25,26,31,34,36 43,51,52,68,69	2	$\boxed{5,6,15,19,21,22,31,36,51,52,68,69}$
3	28,29,41,42	3	$\boxed{17,18,20,23,24,25}$
4	27,30,32,33,35	4	$\boxed{16,26,28,29,34,41,42,43}$
5	37,38,39,40,44,45,47,48 50,53,54,55,59	5	$\boxed{27,32,33}$ $\boxed{30,35}$ $\boxed{44,50,54}$
6	64,65,66,67	6	$\boxed{37,38,40}$ $\boxed{39,47}$
7	46,49,56,57,58,60,61,62,63	7	$\boxed{46,49,56,58,64,65,66,67}$
8	3,4	8	$\boxed{48,59,61}$ $\boxed{57,60,62,63}$
	Partition suggested by Tung [1979]		Solution generated by the high-density technique.

Figure 16

Tung's [1979] 69 node (weighted)

Printed Circuit Board Test Facility design problem

33,35} because only one of them (node 44) is linked to node 33, which in turn is not highly linked. Thus, the density on the arc between 33 and 44 is relatively high because $|N_{33} \cup N_{44}|$ is small.

Herein lies a significant feature of the high-density partitioning technique. A node such as node 45 (Good Printed Circuit Board Simulator) seems to be a focal requirement for a PCB Test Facility, as it is related to 25 percent of the other requirements for the system by the concensus of the designers. Yet node 45 is a low-density node, in the sense that it is incident to arcs of relatively low density. Thus, rather than having node 45 exert a significant influence in the formation of the partition as it might using Huff's approach, it becomes a "fringe" node in the high-density partitioning approach and does not appear in the high-density clusters.

This strategy is appropriate if a node such as node 45 represents a requirement that serves as a "cover-term" for several subtasks in the design problem. In the example above there may be four or five components of the single requirement "Good PCB Simulator." The system designers should be alerted to the existence of such a node, in order to best coordinate work among the components of this specific task.

6.5 The Budgeting System Design Problem

Huff [1979] applied the entire SDM to the problem of designing a budgeting system for M.I.T. He identified 77 functional requirements and 289 weighted interdependencies to form the system design graph, and used his INTERCHANGE algorithm to produce the partition shown in Figure 17. Huff reports no direct computational experience with the 77 node graph; however, he does indicate that the INTERCHANGE algorithm implemented in PL/I on the IBM/370 168 required approximately 9 CPU seconds to decompose a 40-node graph. Since the INTERCHANGE algorithm requires $O(|N|^3)$ operations, it is quite safe to assume that the 77-node graph requires at least 18 CPU seconds.

Figure 17 also shows the high-density partition which required only one second of CPU time to produce. Comparing the results, the high-density partition seems to have some clear superiorities. Huff includes node 27 with {5,6,35} even though it is linked to none of them. The high-density solution joins node 27 with {53,54,55,67,69,70} because of its link to node 53. The clear superiority exhibited by the high-density approach, however, is the speed, with which the technique decomposes the design graph.

Figure 17 here

Figure 18 summarizes the computational performance of the high-density technique for the five design graphs described above and three other graphs taken from the literature. The total time is broken down into two components, calculation of the arc densities and calculation of the clustering and partition trees, in order to demonstrate the computational dominance of the first.

Figure 18 here

1	7,28,38,56,57,58,59,60,61 62,65,66,68,71,76
2	18,19,20,21,22,23,24,25,26 29,31,32,33,34,36,39,40 41,42
3	5,6,27,35
4	11,12,14
5	16,43,44,45,46,47,48,49,50 51,52,64,74
6	15,77
7	9,10,13
8	8,63,75
9	1,2,3,4,17,30,37
10	72,73
11	53,54,55,67,69,70

1	7,58,59,60,61,62,65,66,68,71,76,44 28,56,57
2	18,24,25,26,33,41 20,23,36 21,29,39,40
3	5,6,22,35 11,12,14 31,32,34
4	16,49,51,52 45,47,48,50
5	15,77 42,43,46,64,74
6	8,63,75 9,10,13,38
7	1,2,3,4,17,19,30,37 72,73
8	27,53,54,55,67 69,70

Figure 17

Huff's [1979] 77 node (weighted) Budgeting System Design problem

DATASET	|N|	|A|	d	(A)	(B)
ANDREU DBMS	22	39	3.55	.02	.03
HUFF DBMS	22	39	3.55	.02	.03
SV 25-NODE*	25	44	3.52	.02	.03
AIRPORT	27	96	7.11	.06	.08
SV 51-NODE**	51	126	4.94	.08	.13
TUNG PCB	69	203	5.88	.30	.55
SPHERE ***	74	245	6.62	.30	.51
HUFF BUDGET	77	289	7.51	.33	.66
INDUSTRIAL #	200	466	4.66	.37	.91
IND.+ BANKS #	250	889	7.12	.52	1.71

(A) CPU SECONDS REQUIRED TO CALCULATE BOTH CLUSTERING TREE AND PARTITION TREE

(B) CPU SECONDS REQUIRED TO CALCULATE ARC DENSITIES

* Sangiovanni-Vincentelli et. al. [1977]
** Sangiovanni-Vincentelli et. al. [1977]
*** Levine [1972]
# Lattin [1981]

Figure 18

Computational Results

7. CONCLUSION

We have presented a graph-partitioning technique based on a high-density clustering model. The technique requires no information regarding the number of subgraphs in the decomposition, and utilizes no "goodness-of-partition" measure which might bias the structure of the outcome. The partitioning technique divides the graph into high-density regions, and does so by forming maximum spanning trees. The order of work required to calculate these densities is $O(|A|^2/|N|)$, and the amount of work required to form each spanning tree is proportional to $|A|$. Thus, the technique exploits the sparsity typical of design graphs and provides the system designer with greater computational efficiency.

We also presented evidence of the several advantages of the high-density partitioning technique by comparing its performance against other existing methods. For each of five different design graphs taken from the literature, the solution generated by the high-density partitioning technique was comparable to the partition suggested by the existing decomposition methods. The speed of the new technique indicates that it may be useful for partitioning design problems too large for either McCormick's BEA or Huff's INTERCHANGE.

Further research in support of the design process should focus on the "sensitivity" of the structure of the partition to the specifications made by the system designers. As design graphs become larger, it becomes less and less desirable to partition every possible version of the graph suggested by the system designers. There should be, built into the high-density partitioning technique, some form of diagnostic to alert the designers to changes that might occur in the structure of the partition due to some small perturbation of their specifications. For example, in the 13 node graph presented above, any differential weighting on the links to node 8 would determine its membership in one of the subgraphs, and it would no longer be a "toss-up" node. Such a diagnostic, built into the partitioning method, would provide the designer with a flexible, interactive system to examine the implications of his specifications, look for counter-intuitive results, and further refine his design.

REFERENCES

Andreu, R.C. and S.E. Madnick [1977] "A Systematic Approach to the Design of Complex Systems: Application to DBMS Design and Evaluation," CISR Research Report #32, MIT, Sloan School of Management.

Andreu, R.C. [1977] "Solving Decomposition Problems: Alternative Techniques and Description of Supporting Tools," CISR Technical Report #2, MIT, Sloan School of Management.

Golden, B. and T.L. Magnanti [1982] Network Optimization, forthcoming.

Hartigan, J.A. [1975] Clustering Algorithms. New York: John Wiley.

Huff, S.L. and S.E. Madnick [1978] "An Extended Model for a Systematic Approach to the Design of Complex Systems," CISR Technical Report #7, MIT, Sloan School of Management.

Huff, S.L. [1979] "A Systematic Methodology for Designing the Architecture of Complex Software Systems," Unpublished Ph.D. Dissertation, MIT, Sloan School of Management.

Lattin, J.M. [1981] "Partitioning the Corporate Network," Presented to the Annual Meeting of the Classification Society in Toronto, June 1981.

Levine, J.H. [1972] "The Sphere of Influence," American Sociological Review, v. 37, pp. 14-27.

McCormick, W.T., P.J. Schweitzer, and T.W. White [1972] "Problem Decomposition and Data Reorganization by a Clustering Technique," Operations Research, v. 20, No. 5, pp. 993-1007.

Ross, G.J.S [1969] "Minimum Spanning Tree, Algorithm AS12," Applied Statistics, v. 18, pp. 103-104.

Sangiovanni-Vincentelli, A., L. Chen, and L.O. Chua [1977] "An Efficient Heuristic Cluster Algorithm for Tearing Large-Scale Networks," IEEE Transaction on Circuits and Systems, v. CAS-24, No. 12, pp. 709-717.

Tung, Pei-Ti [1980] "A Systematic Approach to Complex System Design: An Application to Printed Circuit Board Test System Design," CISR Technical Report #13, MIT, Sloan School of Management.

Wong, M.A. [1980] "A Graph Decomposition Technique Based on a High-Density Clustering Model on Graphs," CISR Technical Report #14, MIT, Sloan School of Management.